

GWT From Scratch – Day 5

Building A Web Site From Scratch

The aim of today's session is, like it says, to build a web site from scratch.

You will

- Learn about history tokens and HistoryListeners
- Build a page-creation routine
- Create a tabbed menu (OK, a TabPanel with 3 tabs)
- Style it
- Learn how to write to the history stack
- Learn how to get history events
- See how to react to these events
- Create one of those fade-out-the-screen messageboxes
- Use it to react to broken links and bookmarks
- Learn the Most Important Thing In Web Design (Yep, really - according to me, anyway)
- Get all the code for the whole thing
- Finish the course with a web site - just

Questions?

My ideal is to have no questions at all because everything is perfectly clear, but if you have any questions, then please contact me. I would like to consider the course 'complete' within the limits I have set for it. In other words, I am here to supplement the course if, in any way, it is not well enough explained to get everyone through it. I intend to use the feedback to improve the course and reduce the questions. So any questions are very welcome.

Any Problems

rx01-day4@examples.roughian.com

© Ian Bambury 2008

Creating The Site

I'll use the NewApp project as a basis, as always. But there's nothing at all complicated converting this to what if you have at hand.

We want to be responding to History events, so we need to declare our Main class as a HistoryListener.

You could of course declare it anywhere you like.

The HistoryListener interface demands that we have an onHistoryChanged method which takes as its argument a string containing the historyToken (more of which later).

```
package com.roughian.newapp.client;

import com.google.gwt.core.client.EntryPoint;
import com.google.gwt.user.client.HistoryListener;

public class Main implements EntryPoint, HistoryListener
{
    public void onModuleLoad()
    {
    }

    public void onHistoryChanged(String historyToken)
    {
    }
}
```

The onHistoryChanged method gets called when the user presses the Back or Forward buttons on the browser, or when this is done programmatically, or when we set the history token ourselves in the code.

Although the Main method is now all set up as a HistoryListener, it won't get called when history changes because we haven't let the system know that that is what we want. We do it like this:

```
public class Main implements EntryPoint, HistoryListener
{
    public void onModuleLoad()
    {
        History.addHistoryListener(this);
    }
    public void onHistoryChanged(String historyToken)
    {
    }
}
```

Now, every time the history token changes, the onHistoryChanged method will get called with the new history token. But what is a historyToken?

Well the historyToken tells you what state to put your application into. The only reason it is there in the first place is because you set it yourself.

The History Token Works Like This

When you put your application into a certain state (for example when you display the Home page), you write a historyToken out with something useful like 'Home'.

If a user goes away and then presses the back button to come back to your home page, then your onHistoryChanged routine will be passed the token 'Home'. You have to react to this in the onHistoryChanged routine by changing your application so that it shows the Home page.

It is a good idea to use human-readable tokens because this token will appear after the '#' in the address box of the browser. Thus, it will be in any URL that a visitor decides to send to someone else.

There Is One Slight Complication.

The onHistoryChanged routine gets called every time the history token changes - not a problem you might think.

Let us say that you are on the Home page. You then click on a tab and go to the Products page. At some point when you're showing the Products page, you write out a historyToken.

Because you write out a historyToken, the onHistoryChanged routine gets called. Because JavaScript is single-threaded, this happens after everything your application is doing at the moment (i.e. showing the Products page).

So, once the Products page has been shown, the onHistoryChanged routine will get called with the token 'Products'.

The onHistoryChanged routine, having been called with the token 'Products', will now show the Products page!

While it is showing the Products page, it will write out a historyToken of 'Products'.

Luckily, when it does this, the historyToken doesn't actually change (it stays the same) so the onHistoryChanged routine won't get called a third time, but your application will show the products page twice.

The answer, of course, is to save the historyToken every time the routine is run, and compare the old historyToken with the new history token. If they are the same you exit immediately.

There is no way round this. It's a Catch-22 situation: you have to set the historyToken when you display the Products page or there won't be any history, but because you change the historyToken, the onHistoryChanged routine will get called.

In a program like this, displaying the page twice would not be a problem in. But if you are displaying a complicated UI, you really don't want it run twice, especially if there are RPC calls to the server.

A Page Routine

It is very useful to have a common class for all your pages. It makes menuing systems so much easier because the menu items really don't want to care about all the different kinds of pages you might have, they just want to deal with a generic page, so it's worth coming up with a base class for a generic page for them.

The menu as provided by GWT is the kind of thing you'd expect on a right-click menu, or the drop-down menu systems in almost every desktop application. If you're trying to build an online replacement for a desktop application - something like a word processor - or if you want a right-click menu, then they are excellent.

For a modern website however, they don't really hack it. They look old-fashioned, out of place, and awkward. Websites these days are more streamlined and users don't expect to have to faff around with fiddly little menus like that. The trend is to go for nicer-looking pages, and a wizarddy set-up for things that require some kind of flow. (Although it often gets taken much too far, and people who know what they are doing end up screaming with frustration because they keep having to pass a page that offers an option which they never, ever are going to use - Microsoft please note that there are people out there who know what they are doing, and not all of us need an animated paperclip telling us that it looks like we are writing a letter - I know when I'm writing a bloody letter).

So here is a simple little page routine.

```
FlowPanel page(String html)
{
    FlowPanel panel = new FlowPanel();

    HTML page = new HTML(html);
    panel.add(page);

    page.setSize("300px", "200px");
    page.setHorizontalAlignment(HasAlignment.ALIGN_CENTER);

    DOM.setStyleAttribute(page.getElement(),
        "border", "10px solid blue");
    DOM.setStyleAttribute(page.getElement(),
        this "padding", "20px");

    return panel;
}
```

The FlowPanel we mentioned before it is just an ordinary div which allows elements to flow in the expected way.

The routine simply takes the String that you pass to it, and returns a formatted FlowPanel.

We are going to put this FlowPanel into a DeckPanel (which is what the bottom half of a TabPanel is). Within a TabPanel, we have a problem, because the TabPanel sets the width and height of anything you put into the DeckPanel to 100%.

This means if you want to put any kind of border or margin on it then, if the browser is working to CSS standards, what you put into the DeckPanel is now 100% plus the border times two, plus the margin times two.

What this means is, in effect, that part of whatever you put into the DeckPanel will be hidden if you add a border or a margin.

Back to our program. In these 'pages', all we want to do is show a little bit of text so we can tell that the page has changed.

The Menu

For the menuing system, we are simply going to have a TabPanel. The code looks like this:

```
String email = "ian@examples.roughian.com";
TabPanel menu = new TabPanel();

public void onModuleLoad()
{
    RootPanel.get().add(menu);

    menu.add(page("Welcome to my Home Page"), "Home");
    menu.add(page("This page is all about Me"), "About Me");
    menu.add(page("If you click <a href='mailto:" + email + "
'>"
        + "here</a> you can email me"), "Contact Me");

    menu.selectTab(0);

    History.addHistoryListener(this);
}
```

A TabPanel with three 'pages' in it.

The first thing we do in the onModuleLoad method is to add the menu to the RootPanel. It's just a safety thing with me, remember? Heights and widths, left and top, most of the time you will get a problem with these if the widget is not attached to the RootPanel. Okay, so we don't use them here, but if I make it a habit, it means that I don't forget.

Then we add three pages, just dummy ones here, with a little bit of content. To do this, we add a tab to the TabPanel.

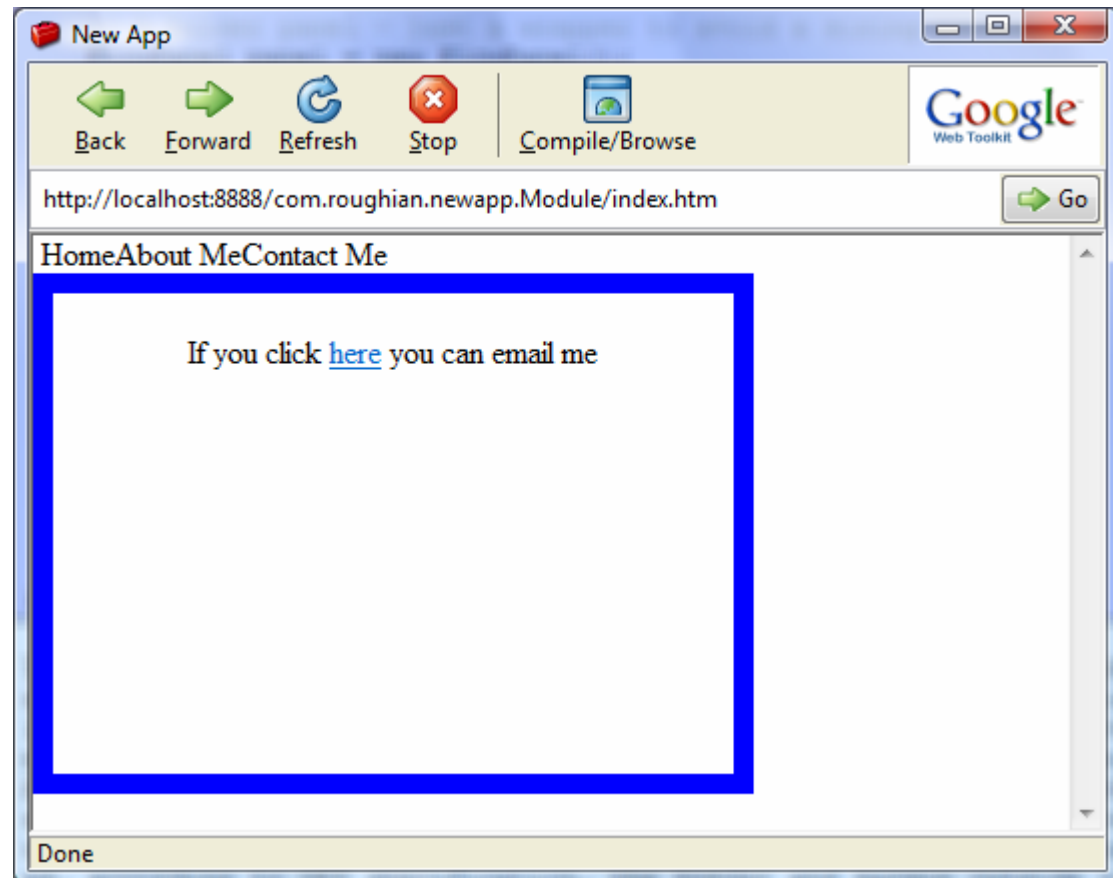
The first parameter that is passed to a TabPanel when we create a new tab, is whatever you want to go into the bottom part to be displayed when the tap is active. In this case it is the FlowPanel which is page.

The second parameter after the page (for example 'Home') is what will be displayed in the tab.

Then we select the first tab which has an index of zero. This is in effect then, our default page.

Lastly in this piece of code we add this class as a HistoryListener which ensures that 'this' will get notified of history events.

So finally we have something we can actually run. And it looks sort of okay. Except that the TabPanel is missing.



I'll go and get the CSS that we used yesterday when we were styling the TabPanel, and add it in. Here it is:

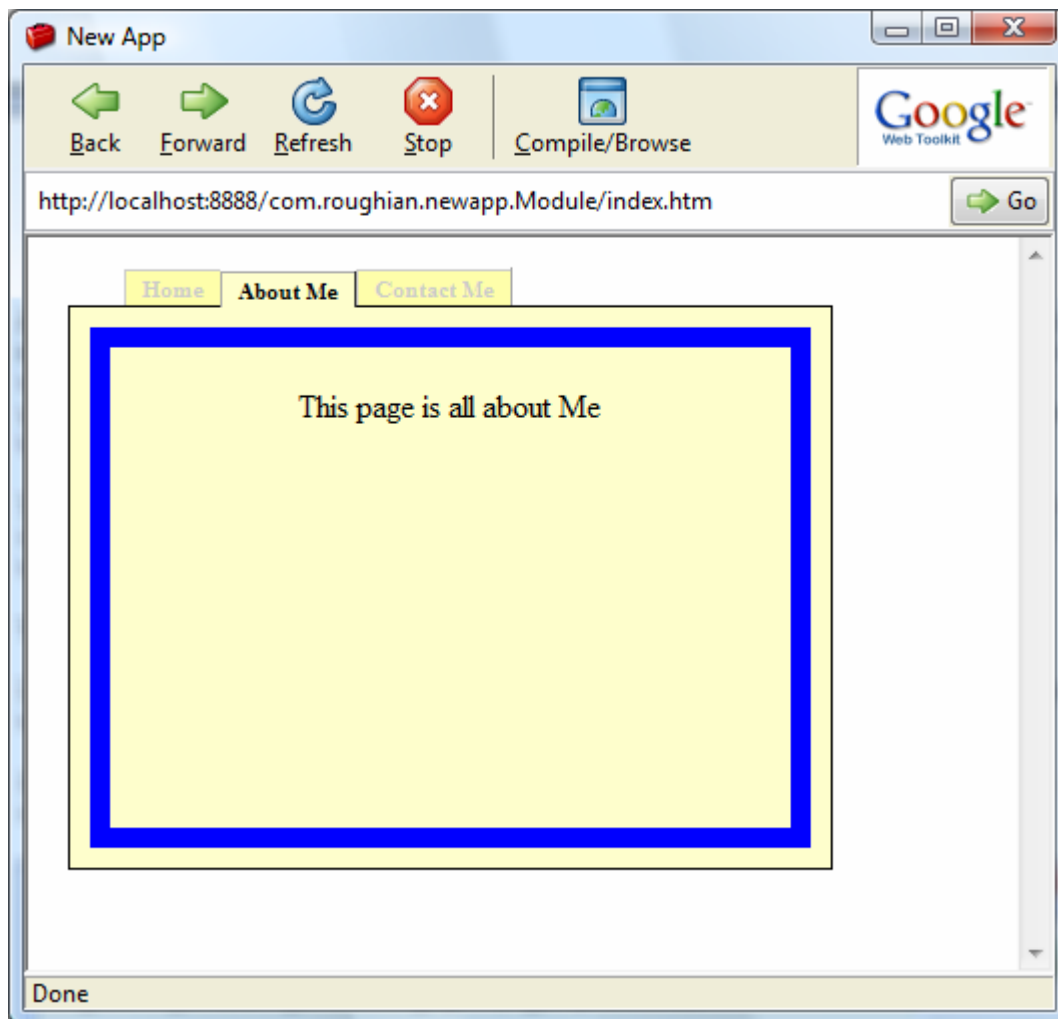
```
/* *****
*                               TabPanel
* ***** */
.gwt-TabPanel
{
    margin                :    20px;
}
.gwt-TabPanelBottom
{
    background-color      :    #ffc;
    border-left           :    1px solid black;
    border-right          :    1px solid black;
    height                :    98%;
}
```

```

        border-bottom      :    1px solid black;
        padding            :    10px;
    }
    .gwt-TabBar
    {
        margin-top         :    15px;
        height             :    100%;
    }
    .gwt-TabBar .gwt-TabBarFirst
    {
        height             :    100%;
        width              :    25px;
        padding-left       :    3px;
        border-bottom      :    1px solid black;
    }
    .gwt-TabBar .gwt-TabBarRest
    {
        border-left        :    1px solid #AAA;
        border-bottom      :    1px solid black;
        padding-right      :    3px;
    }
    .gwt-TabBar .gwt-TabBarItem
    {
        border-top         :    1px solid #ccc;
        border-left        :    1px solid #ccc;
        border-bottom      :    1px solid black;
        background-color   :    #ffa;
        color              :    #ccc;
        font-size          :    70%;
        font-weight        :    bold;
        padding            :    2px 8px 2px 8px;
        cursor             :    pointer;
        cursor             :    hand;
    }
    .gwt-TabBar .gwt-TabBarItem-selected
    {
        color              :    black;
        background-color   :    #ffc;
        border-top         :    1px solid #aaa;
        border-left        :    1px solid #aaa;
        border-right       :    1px solid #333;
        border-bottom      :    0px solid black;
        padding            :    2px 8px 2px 8px;
        cursor             :    default;
    }
}

```

And this is what it does for our miniature website.



We are pretty much there now, but we are missing the history support.

History Support

In the `onHistoryChanged` routine, we need to react to the history being changed (obviously).

What you can do, and what I do, is to make this routine your main control routine for displaying the various pages and/or states for your application choose one.

All you do is to call `Main.onHistoryChanged("WhateverPage")`.

Writing History Out

But as I'm sure you know by now, we can't react to any history events yet, because we haven't written out any history records. Right now if the user presses the Back button in the browser then they will go back to the page they were out before they came to our site.

TabPanels can have TabListeners. These have two methods: `onBeforeTabSelected` and `onTabSelected`. If you use Eclipse's autocomplete feature, then be wary, it

defaults the return of `onBeforeTabSelected` to false which cancels the changing of the tab.

What we want to do, is to write a new item to the history stack. We do it like this:

```
public void onTabSelected(SourcesTabEvents sender, int
tabIndex)
{
    TabPanel panel = (TabPanel)sender;
    String text = panel.getTabBar().getTabHTML(tabIndex);
    History.newItem(text);
}
```

We're not using the `onBeforeTabSelected` method, so we just return true (i.e. we want the new tab to be displayed).

In the `onTabSelected` method we first cast the sender as a `TabPanel` and refer to it as 'panel'.

In the next line, we get the text from the tab bar. So, from the panel we get the `tabBar`, and from the `TabBar` we get the HTML of the current tab (since the tab has been selected, this is now the one we want to write the history for), and we have been passed the `tabIndex` so we know which one it is.

All that is left to do, is write the history item out.



And there it is working. But there are a couple of problems. The first one is that when the program first starts, we go to the first tab.

The problem of it doing that may not at first be apparent. But consider what happens if somebody has bookmarked your About Me page and then goes back to the bookmark. They don't want the home page, which is what they are going to get.

We have to send them to where they want to go and we do this with the following line:

```
History.onHistoryChanged(History.getToken());
```

The other problem is that the About Me bookmark has the ugly %20 in it. It makes it much less readable. So what I suggest we do is change it to an underscore.

When we write out history we replace spaces with underscores. Where we search for the tab, we replace underscores with spaces. The only problem is that you can't use underscores in your tab names. I can live with that more than with %20.

```
History.newItem(text.replaceAll(" ", "_"));
```

Reacting To History Change

We now have to react to history change. Here's the code:

```
String oldToken = null;

public void onHistoryChanged(String historyToken)
{
    if(oldToken != null && historyToken.equals(oldToken))
    return;
    oldToken = historyToken;

    int index = 0;

    TabBar tabbar = menu.getTabBar();

    for(int i = 0; i < tabbar.getTabCount(); i++)
    {
        if(tabbar.getTabHTML(i)
            .equals(historyToken.replaceAll("_", " ")))
        {
            index = i;
            break;
        }
    }

    menu.selectTab(index);
}
```

Just to run you through it.

We have oldToken to store the old token in.

If the old token is not equal to null (this will be the case the first time through) and then if the historyToken is the same as the old token, then there is nothing to do because we are showing the right page. So we exit immediately.

We default the index to zero (i.e. show the first tab).

For readability reasons we can get the TabBar and assign it To a variable.

We then run through each of the tabs in turn looking for a match between what is written on the tab and what is in historyToken (remembering, of course, to replace spaces with underscores).

If we find it, we assign the variable to the index we are going to use to select a tab with, and then break out of the for loop.

Once the loop has ended (either because we found it and broke the loop, or because we fell through the loop and the index has retained his value of zero) we then select the tab.

Done!

And there you have it a working three-page web site.

One Step Beyond

One Step Beyond, surely that's Madness? Yep, 1979 debut album. (Yes, I am that old).

I did promise you one of those pop-ups where the background is all faded out. And there's a good use for it here. (Well sort of.)

Consider what would happen if you change the Contact Me to Contact Us. If someone had bookmarked the Contact Me page, then you changed it's name, and then they followed that bookmark - what would happen?

They would go to the Home page.

It would be better if your site popped up a message to say that you couldn't find that page, sorry. So we will do that. The first thing we need to do is to create a Blinded Pop-Up.

Blinded Pop-Up

What we will do is create a reusable class that you can pop up from anywhere and it will automatically first put up an opaque panel over the whole screen, and then pop up the message. Making sure, of course, that we get rid of the opaque panel when the message disappears.

There is, of course, a problem - I like problems, if it were easy and everyone can do it, then no-one would pay me lots of money for doing what I consider a hobby. Anyway, the problem is this:

The scrollbar of a browser is part of the browser. Not surprising, I hear you say, but it is a problem nevertheless. It is basically a part of the BODY tag, but your HTML is inside the BODY tag and so anything that you try to pop-up, even at maximum, will not cover the BODY tag's scrollbar because it will be inside the BODY tag.

The way around this is to turn scrolling off in the BODY tag, and have a div the same size as the BODY tag and make that scroll. I give this the ID of 'screen'.

If you want your website be less than the full width of the screen (for example you might want it to be eight hundred pixels wide and centred on the screen) then you will need another div inside of that. I call this one 'site'. It is well worth having this 'site' div and setting some kind of limit for the width because if you have a fairly wide screen like I do, you get to realise how naff some sites can look (and how unreadable they become) at high resolutions.

This may seem a lot of trouble a look at it like this: when I did it, I only had to do it once and ever since then I have copied a basic project and so the effort I put in has been spread amongst all the projects I have done ever since. For you, it is even easier because all you have to do is copy my project - and for me that is good as well, because the effort I put in to get the darn thing going a spread even more thinly.

It seems nice and easy when someone just write down a sequence all the things you have to do to create a site like this, but you don't realise how many dead ends I went down to get here :-)

Okay so here's the HTML:

```
<body>
  <div id="screen">
    <div id="site">
    </div>
  </div>
</body>
```

Yes I know what you're thinking, how many dead ends could you possibly go down when you're writing something like this? You have no idea how many other things I tried :-) You'll need some CSS as well:

```
body,html
{
    height          :    100%;
    width           :    100%;
    overflow        :    hidden;
}
#screen
{
    height          :    100%;
    width           :    100%;
    overflow        :    scroll;
    /* stop absolutepanels failing to scroll */
    position        :    relative;
}
```

You need to set the body and html to 100%. You need to set the overflow to 'hidden' to stop the body tag from scrolling.

We then set the 'screen' div two 100% also. '100%' can cause problems for a lot of people in other situations, but here the HTML tag takes an absolute size from the browser window, the BODY tag takes it from the HTML and the 'screen' tag takes it from BODY tag. There is a direct line of descendance. We set the overflow to 'scroll', and then we set the position to 'relative' to stop a bug which I'm not going to explain here.

I'll give you the complete code at the end so don't worry that you are going to have to pieces altogether later.

Okay, so we set the background, and now we can move on to the pop-up itself. We start by building a class, declaring the background pop-up (which I given an ID of 'glass'), and sizing it so it will fit the whole screen.

```
class BlindedPopup
{
    public BlindedPopup(String message, boolean
addDefaultStyles)
    {
```

```

        glass = new PopupPanel();
        DOM.setStyleAttribute(glass.getElement(), "width",
"100%");
        DOM.setStyleAttribute(glass.getElement(), "height",
"100%");
    }
}
The next thing to do is to add in the message pop-up.
class BlindedPopup
{
    PopupPanel glass;
    PopupPanel popup;

    public BlindedPopup(String message, boolean
addDefaultStyles)
    {
        glass = new PopupPanel();
        DOM.setStyleAttribute(glass.getElement(), "width",
"100%");
        DOM.setStyleAttribute(glass.getElement(), "height",
"100%");

        popup = new PopupPanel(false);
    }
}

```

After that, we need a label for the header of the pop-up, and HTML widget for the main message, and a button so that the user can close it.

The pop-up which it is based on a SimplePanel. A SimplePanel can only hold one widget. What we can do though, is to add a single widget to the simple panel which can itself hold more than one widget. This is allowable. For this I have used a VerticalPanel.

We nearly have to show a glass pop-up since it already knows that it has to cover the screen. The main pop-up needs to be centred and the popup.center() not only centres the pop-up, but it also shows it.

In the code that follows, you should be to understand everything in it now.

```

class BlindedPopup
{
    PopupPanel glass;
    PopupPanel popup;

    public BlindedPopup(String message, boolean
addDefaultStyles)
    {
        glass = new PopupPanel();
        DOM.setStyleAttribute(glass.getElement(), "width",
"100%");
        DOM.setStyleAttribute(glass.getElement(), "height",
"100%");

        popup = new PopupPanel(false);
    }
}

```

```

        VerticalPanel popupContents = new VerticalPanel();
        Label header = new Label("BlindedPopup Demo");
        HTML html = new HTML(message);
        Button closeButton = new Button("Close");

        popup.add(popupContents);
        popupContents.add(header);
        popupContents.add(html);
        popupContents.add(closeButton);
        popupContents.setCellHorizontalAlignment(
            closeButton, HasAlignment.ALIGN_CENTER);
        glass.show();
        popup.center();
    }
}

```

The only thing missing, is the ClickListener for the button.

```

public void onClick(Widget sender)
{
    popup.hide();
    glass.hide();
}

```

We need some CSS to make it look right.

```

/*****
*          Blinded Popup
*****/
.rx-popup
{
    background-color      :    #ffc;
    border                :    3px solid #009;
}
.rx-popup-header
{
    background-color      :    #ff0;
    font-size             :    90%;
    font-weight           :    bold;
    border-bottom         :    3px solid #009;
    padding               :    5px;
    text-align             :    center;
}
.rx-popup-message
{
    font-size             :    80%;
    padding               :    15px;
}
.rx-popup-footer
{
    padding               :    5px;
    text-align            :    center;
    width                 :    100%;
}

```

```
.rx-glass
{
    background-color      :    #000;
    opacity               :    0.70;
    -moz-opacity          :    0.70;
    filter                :    alpha(opacity=70);

    width                 :    100%;
    height                :    100%;
}
```

We now have a working pop-up which blinds out the background. Here's the whole class.

```
class BlindedPopup implements ClickListener
{
    PopupPanel popup;
    PopupPanel glass;
    public BlindedPopup(String header, String Message)
    {

        glass = new PopupPanel();
        glass.setStyleName("rx-glass");

        DOM.setStyleAttribute(glass.getElement(),
            "width", "100%");
        DOM.setStyleAttribute(glass.getElement(),
            "height", "100%");

        popup = new PopupPanel(false);
        popup.setStyleName("rx-BlindedPopup");

        Label header = new Label("BlindedPopup Demo");
        header.setStyleName("rx-BlindedPopup-header");

        HTML html = new HTML(message);
        html.setStyleName("rx-BlindedPopup-message");

        Button closeButton = new Button("Close", this);

        popupContents.add(header);
        popupContents.add(html);
        popupContents.add(closeButton);
        popupContents.setCellHorizontalAlignment(closeButton,
            HasAlignment.ALIGN_CENTER);
        popup.add(popupContents);

        glass.show();
        popup.center();
    }
    public void onClick(Widget sender)
    {
        popup.hide();
        glass.hide();
    }
}
```



```
}
```

Now we need to change the HistoryListener so that it recognises a duff historyToken.

```
public void onHistoryChanged(String historyToken)
{
    if(oldToken != null && historyToken.equals(oldToken))
return;

    int index = -1;
    if(historyToken.equals("")) index = 0;

    oldToken = historyToken;

    TabBar tabbar = menu.getTabBar();

    for(int i = 0; i < tabbar.getTabCount(); i++)
    {
        if(tabbar.getTabHTML(i)
            is.equals(historyToken.replaceAll("_", " ")))
        {
            index = i;
            break;
        }
    }

    if(index == -1)
    {
        new BlindedPopup(
            "Sorry, There's A Problem",
            "The page you requested has moved");
        index = 0;
    }
    menu.selectTab(index);
}
```

The difference here is that we now set the index to -1 as an indicator that we haven't found the correct page yet. Doing this is on the edge of bad programming practice - we should really use a boolean for that since we are giving that variable two uses, but we can get away with it because it is so widespread.

If the historyToken is empty, that almost certainly means that the visitor has just arrived. So we set the index to zero, which is default page. You could use anything is default page, but zero means that your program will still work even if there is only one page. Anything higher and you just have one more than the index says.

In the greyed-out area we are still running to the tabs looking for the correct page. The difference being now a little section after in bold type: if the index is still -1 then we display a pop-up and set the index to zero to show the default page. Again we are on the edge of bad programming practice, we really shouldn't have 'magic numbers' in our program, we should set a constant to zero, call it `DEFAULT_INDEX_PAGE` and use that instead.

There are two reasons why didn't do that, firstly because it's too much trouble (mind you, it would have been a lot less trouble than it is right now having to explain why I

didn't do it), secondly, because some bugger might try and change it, and it's a lot safer at zero than it is at anything else.

So that's pretty much it then. If you run it, but I don't suppose that you will right now because I don't suppose that you've been cutting and pasting everything into a project all this time. But if you did you would find that it worked as expected, and if you change the bookmark in the address box from, say, 'Home' to 'Homer', and refresh the browser (or hosted browser) then you will get a pop-up has expected.

One More Thing...

Before I sign off, I would like to leave you with The Most Important Thing In Web Development. I know you won't appreciate it, that is what it will do is stop you having a lot of problems, and you won't know the problems you're not having.

You already know that GWT will take all the problems out of cross-browser JavaScript. The only big area left where there could be problems is in your CSS.

The problem is often because the different browsers start off with different default settings. If you get a layout problem, and usually it's only a pixel or two, you look at your CSS and can't see what's wrong. Because it's not in your CSS, it's in the defaults.

So in the final program listing, you will see a section of CSS (which you really ought to break out into a separate file) which will reset all the major browsers to a common level. If I were you, I would put in a section between that and the next section where you set up all the default values that you yourself would like (for example, a default text colour).

Another More Thing...

For homework, you can work out a way so that if a page has moved, the user gets redirected transparently from the old bookmark to the new bookmark.

Thank You

Thank you for sticking with me through this course, I hope you got something out of it - I expect you did for no other reason than the fact that you are reading this.

The autoresponder will be sending out a request for comments a few hours after the email giving you this link was sent out. The course is free, but it would be very helpful to me if you could find a moment to send me an email with a comment or two about the course.

I'm not necessarily looking for praise (nice though it is), what are very useful are comments telling me how it can be improved. If there are bits of the text that don't make any sense at all, then please let me know. It might just be because I'm using a speech recognition system to type this, and it's not bad, but it's not perfect.

Okay, here's the complete code (Java and HTML), and then I'm out of here.

Regards,

Ian

<mailto:ianbambury@gmail.com>

index.htm

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
  <head>
    <title>New App</title>
    <meta http-equiv="Content-Type" content="text/html;
charset=utf-8">
    <script language="javascript"
src="com.roughian.newapp.Module.nocache.js"></script>

    <!-- link type='text/css' rel='stylesheet'
href='gwt.css' / -->
    <style>

/*****
*           Levelling
*****/
body,div,dl,dt,dd,ul,ol,li,h1,h2,h3,h4,h5,h6,pre,form,fieldset
,input,textarea,p,blockquote,th,td
{
    margin                :    0;
    padding               :    0;
}
table
{
    border-collapse       :    collapse;
    border-spacing        :    0;
}
fieldset,img
{
    border                :    0;
}
address,caption,cite,code,dfn,em,strong,th,var
{
    font-style            :    normal;
    font-weight           :    normal;
}
ol,ul
{
    list-style            :    none;
}
caption,th
{
    text-align            :    left;
}
h1,h2,h3,h4,h5,h6
{
    font-size             :    100%;
    font-weight           :    normal;
}
q:before,q:after
{
```

```

        content                :    '';
    }
abbr,acronym
{
    border                    :    0;
}
/*****
*           My Defaults
*****/
body, h1, h2, h3, h4, h5, h6, dl, dt, dd, ul, ol, li, p, th,
td, form, fieldset, input, textarea, blockquote
{
    font-family                :    Arial, Verdana, Tahoma,
    Sans-Serif;
    color                      :    #009;
}
h1        {    font-size      :    250%;}
h2        {    font-size      :    150%;}
h3        {    font-size      :    120%;}
h4        {    font-size      :    100%;}
h5        {    font-size      :    80%;}
h6        {    font-size      :    70%;}

p         {    font-size      :    80%;}
.quote    {    font-size      :    80%;}
button    {    font-size      :    80%;}

pre
{
    background-color           :    #ccf;
    margin                     :    10px;
    padding                    :    10px;
    border                     :    1px solid #000;
    font-size                   :    10pt;
}
/*****
*           Project Setup
*****/
body,html
{
    height                     :    100%;
    width                      :    100%;
    overflow                   :    hidden;
}
#screen
{
    height                     :    100%;
    width                      :    100%;
    overflow                   :    scroll;
    /* stop absolutepanels failing to scroll */
    position                   :    relative;
}
/*****
*           Blinded Popup
*****/

```

```

.rx-popup
{
    background-color          :    #ffc;
    border                    :    3px solid #009;
}
.rx-popup-header
{
    background-color          :    #ff0;
    font-size                 :    90%;
    font-weight               :    bold;
    border-bottom             :    3px solid #009;
    padding                   :    5px;
    text-align                :    center;
}
.rx-popup-message
{
    font-size                 :    80%;
    padding                   :    15px;
}
.rx-popup-footer
{
    padding                   :    5px;
    text-align                :    center;
    width                     :    100%;
}
.rx-glass
{
    background-color          :    #000;
    opacity                   :    0.70;
    -moz-opacity              :    0.70;
    filter                    :    alpha(opacity=70);

    width                     :    100%;
    height                    :    100%;
}

/*****
*           TabPanel
*****/
.gwt-TabPanel
{
    margin                    :    20px;
}
.gwt-TabPanelBottom
{
    background-color          :    #ffc;
    border-left               :    1px solid black;
    border-right              :    1px solid black;
    height                    :    98%;
    border-bottom             :    1px solid black;
    padding                   :    10px;
}
.gwt-TabBar
{
    margin-top                :    15px;
}

```

```

        height                :    100%;
    }
    .gwt-TabBar .gwt-TabBarFirst
    {
        height                :    100%;
        width                  :    25px;
        padding-left           :    3px;
        border-bottom          :    1px solid black;
    }
    .gwt-TabBar .gwt-TabBarRest
    {
        border-left            :    1px solid #AAA;
        border-bottom          :    1px solid black;
        padding-right          :    3px;
    }
    .gwt-TabBar .gwt-TabBarItem
    {
        border-top             :    1px solid #ccc;
        border-left            :    1px solid #ccc;
        border-bottom          :    1px solid black;
        background-color       :    #ffa;
        color                   :    #ccc;
        font-size              :    70%;
        font-weight            :    bold;
        padding                :    2px 8px 2px 8px;
        cursor                  :    pointer;
        cursor                  :    hand;
    }
    .gwt-TabBar .gwt-TabBarItem-selected
    {
        color                   :    black;
        background-color       :    #ffc;
        border-top             :    1px solid #aaa;
        border-left            :    1px solid #aaa;
        border-right           :    1px solid #333;
        border-bottom          :    0px solid black;
        padding                :    2px 8px 2px 8px;
        cursor                  :    default;
    }
}

</style>
</head>
<body>
    <iframe id="__gwt_historyFrame"
style="width:0;height:0;border:0"></iframe>
    <div id="screen">
        <div id="site">
        </div>
    </div>
</body>
</html>

```

Main.java

```
package com.roughian.newapp.client;

import com.google.gwt.core.client.EntryPoint;
import com.google.gwt.user.client.DOM;
import com.google.gwt.user.client.History;
import com.google.gwt.user.client.HistoryListener;
import com.google.gwt.user.client.ui.Button;
import com.google.gwt.user.client.ui.ClickListener;
import com.google.gwt.user.client.ui.FlowPanel;
import com.google.gwt.user.client.ui.HTML;
import com.google.gwt.user.client.ui.HasAlignment;
import com.google.gwt.user.client.ui.Label;
import com.google.gwt.user.client.ui.PopupPanel;
import com.google.gwt.user.client.ui.RootPanel;
import com.google.gwt.user.client.ui.SourcesTabEvents;
import com.google.gwt.user.client.ui.TabBar;
import com.google.gwt.user.client.ui.TabListener;
import com.google.gwt.user.client.ui.TabPanel;
import com.google.gwt.user.client.ui.VerticalPanel;
import com.google.gwt.user.client.ui.Widget;

public class Main implements EntryPoint, HistoryListener, TabListener
{
    /*
     * Constants and settings
     */
    private static String email = "ian@examples.roughian.com";
    /*
     * Class-level widgets
     */
    TabPanel menu = new TabPanel();

    /*
     * Entry point
     */
    public void onModuleLoad()
    {
        /*
         * Add the menu (the whole site, in our case) to the root panel
         */
        RootPanel.get("site").add(menu);
        /*
         * Add pages to the TabPanel
         */
        menu.add(page("Welcome to my Home Page"), "Home");
        menu.add(page("This page is all about Me"), "About Me");
        menu.add(page("If you click <a "
            + "href='mailto:" + email + "'>here</a>"
            + " you can email me"), "Contact Me");
        /*
         * Add a listener to the tab panel
         */
        menu.addTabListener(this);
        /*
         * Make this class responsible for history
         */
        History.addHistoryListener(this);
        /*
         * Set up the site. The token will be the bookmark if there was one, or ""
         * which will give us the default page
         */
        History.onHistoryChanged(History.getToken());
        /*
         * Done
         */
    }

    public boolean onBeforeTabSelected(SourcesTabEvents sender, int tabIndex)
    {
        /*
         * Not used here. Just say 'OK'.
         */
    }
}
```



```

        return true;
    }

    public void onTabSelected(SourcesTabEvents sender, int tabIndex)
    {
        /*
         * Cast the sender to a TabPanel, get the tab's text, and save it to the
         * history stack
         */
        TabPanel panel = (TabPanel) sender;
        String text = panel.getTabBar().getTabHTML(tabIndex);
        History.newItem(text.replaceAll(" ", "_"));
    }

    // A place to save the old token.
    String oldToken = null;

    public void onHistoryChanged(String historyToken)
    {
        /*
         * If the history token matches the last history token, there's nothing
         * to do
         */
        if(oldToken != null && historyToken.equals(oldToken)) return;
        /*
         * Save the old token for next time
         */
        oldToken = historyToken;
        /*
         * -1 means we haven't found it yet
         */
        int index = -1;
        if(historyToken.equals(""))
        {
            /*
             * The historyToken is blank, so we default the first tab
             */
            index = 0;
        }
        else
        {
            /*
             * There's something in the historyToken, so we need to loop through
             * all of the tabs looking for the one we want. The first thing we
             * need to do is get the TabBar from the menu so we can loop through
             * it
             */
            TabBar tabbar = menu.getTabBar();
            /*
             * Now we loop through all of the tabs looking for the one we want.
             */
            for(int i = 0; i < tabbar.getTabCount(); i++)
            {
                /*
                 * If the come tab is home looking for after we have replaced
                 * ... underscores with spaces
                 */
                if(tabbar.getTabHTML(i).equals(historyToken.replaceAll("_", " ")))
                {
                    /*
                     * ... then we know the index of the tab we want and can
                     * break out of the for loop
                     */
                    index = i;
                    break;
                }
            }
        }
        /*
         * If the index is still -1, and we haven't found our tab page...
         */
        if(index == -1)
        {
            /*
             * ... so we put a message to that effect on the screen, and then we
             * set the index to zero so that we get the default page
            */

```

```

        */
        new BlindedPopup("Sorry, There's A Problem",
            "The page you requested has moved");
        index = 0;
    }
    /*
    * We now know which tab within the selected so we can do that. If we
    * put a blinded pop-up up, then this will still keep happening
    * underneath
    */
    menu.selectTab(index);
}

/*
* This is subroutine to produce a tiny little webpage which we can use in
* the demo. In a real situation you would have a base class which allowed
* for lazy-loading, and build on that.
*/
FlowPanel page(String html)
{
    // Create a panel for us to return
    FlowPanel panel = new FlowPanel();

    // Create an HTML widget containing the HTML we have been passed
    HTML page = new HTML(html);

    // Add it to the page
    panel.add(page);

    // Give it some dimensions
    page.setSize("300px", "200px");

    // Align the text
    page.setHorizontalAlignment(HasAlignment.ALIGN_CENTER);

    // Give it a flashy blue border
    DOM.setStyleAttribute(page.getElement(), "border", "10px solid blue");
    DOM.setStyleAttribute(page.getElement(), "padding", "20px");

    // Return the panel to whatever asked for it
    return panel;
}

/*
* A class which pops up first a blinder screen (which is a screen to cover
* the whole of the browser screen with an opaque haze), and then a message
* box.
*/
class BlindedPopup implements ClickListener
{
    /*
    * The pop-up itself (the one with the message), and the glass panel
    * (the opaque binder)
    */
    PopupPanel popup;
    PopupPanel glass;

    public BlindedPopup(String title, String message)
    {
        /*
        * Create the glass panel and set a style name for it
        */
        glass = new PopupPanel();
        glass.setStyleName("rx-glass");

        /*
        * Set the width and the height
        */
        DOM.setStyleAttribute(glass.getElement(), "width", "100%");
        DOM.setStyleAttribute(glass.getElement(), "height", "100%");

        /*
        * Create a message pop-up and give it a style
        */
        popup = new PopupPanel(false);
        popup.setStyleName("rx-popup");
    }

```

```

        /*
        * Create the header label for the message pop-up and give it a
        * style
        */
        Label header = new Label(title);
        header.setStyleName("rx-popup-header");

        /*
        * Create the HTML widget for the actual message and give it a style
        */
        HTML html = new HTML(message);
        html.setStyleName("rx-popup-message");

        /*
        * Create a button for the user to close the pop-up window
        */
        Button closeButton = new Button("Close", this);

        /*
        * Create a VerticalPanel to hold the various parts of the message
        * box we are going to pop up and add it to the pop-up
        */
        VerticalPanel popupContents = new VerticalPanel();
        popup.add(popupContents);
        /*
        * At all the various bits to the panel which will hold the contents
        * of the message box
        */
        popupContents.add(header);
        popupContents.add(html);
        popupContents.add(closeButton);
        /*
        * Is set the horizontal alignment of the cell that holds the close
        * button so that it is centred
        */
        popupContents.setCellHorizontalAlignment(closeButton,
            HasAlignment.ALIGN_CENTER);

        /*
        * Show the glass panel
        */
        glass.show();
        /*
        * Show the pop-up message box
        */
        popup.center();
    }

    /*
    * This method is run on the close button is clicked, and its purpose is
    * to clean up the screen
    */
    public void onClick(Widget sender)
    {
        /*
        * Hide the pop-up and the glass blinder panel
        */
        popup.hide();
        glass.hide();
    }
}
}

```